LAB 3

Objective: Review a useful Docker networking feature and start the basic components of the Kubernetes head node.

This lab is made of two activities:

- Use Docker networking to start containers that share the same network stack (i.e network namespace).
- Start a Kubernetes head node with **etcd**, the API server, and the controller manager.

In both activities, you need to be on a Docker host.

Docker networking

Docker allows you to specify different types of networking for your containers. One type that is very handy and used in Kubernetes is the *container* type. It allows you to share a network between two containers. In this exercise, you will illustrate this with three **`bash`** commands:

On a Docker host, start a **`busybox`** container that just waits. Then, start another container, using the **`--net=container:<other_container_name>`** option.

```
$ docker run -d --name=source busybox sleep 3600
$ docker run -d --name=same-ip --net=container:source busybox sleep 3600
```

Now, check the IP address of each container, something like the following should do:

```
$ docker exec -ti same-ip ifconfig
```

You should see that the second container inherited the same IP address as the *source* container. That is because, using the Docker **`--net`** option, we were able to use the same networking namespace to both containers.

That is what makes a Kubernetes **POD**, a set of containers running on the same host and that share the same networking namespace. This is what Kubernetes calls the *single IP-per-Pod model*: making a Pod almost look like a virtual machine running multiple processes with a single IP address.

Start your first Kubernetes head node

In this exercise, you will create the beginning of a Kubernetes cluster on your local Docker host. You will use Docker images to run **etcd**, the API server, and the controller manager.

In addition, those three components will share the same network, using the technique highlighted in the first exercise. This will show you how to run three processes in a "Pod".

First, you need **etcd**, you can run it like this:

```
$ docker run -d --name=k8s -p 8080:8080 gcr.io/google_containers/etcd:2.2.1
etcd --data-dir /var/lib/data
```

Then, you will start the API server using the so-called `hyperkube` image, which contains the API server binary. We use a few simple settings to serve the API insecurely.

```
$ docker run -d --net=container:k8s
gcr.io/google_containers/hyperkube:v1.5.1 /apiserver
--etcd-servers=http://127.0.0.1:4001 \
--service-cluster-ip-range=10.0.0.1/24 \
--insecure-bind-address=0.0.0.0 \
--insecure-port=8080 \
--admission-control=AlwaysAdmit
```

Finally, you can start the `Admission` controller, which points to the API server.

```
$ docker run -d --net=container:k8s
gcr.io/google_containers/hyperkube:v1.5.1 /controller-manager
--master=127.0.0.1:8080
```

Notice that since **etcd**, the API server, and the controller manager share the same network namespace, they can reach each other on 127.0.0.1, even though they are running in different containers.

To test that, you have a working setup, use **etcdctl** in the **etcd** container, and list the what is the `/registry` directory:

```
$ docker exec -ti k8s etcdctl ls /registry
```

You can of course reach your Kubernetes API server and start exploring the API:

```
$ curl http://127.0.0.1:8080/api/v1
```

Note that in this exercise you did not start the scheduler, nor did you setup nodes with the **kubelet** and the **kube-proxy**. This is also a toy setup and you may want to set things up differently from scratch.